[0030] The JVM 108 generally comprises a combination of software and hardware. The software may include the compiler 110 and the hardware may include the JSM 102. The JVM may include a class loader, bytecode verifier, garbage collector, and a bytecode interpreter loop to interpret the bytecodes that are not executed on the JSM processor 102.

[0031] FIG. 3 shows an exemplary block diagram of the JSM 102. As shown, the JSM includes a core 120 coupled to data storage 122 and instruction storage 130. The core may include one or more components as shown. Such components preferably include a plurality of registers 140, three address generation units ("AGUs") 142, 147, micro-translation lookaside buffers (micro-TLBs) 144, 156, a multi-entry micro-stack 146, an arithmetic logic unit ("ALU") 148, a multiplier 150, decode logic 152, and instruction fetch logic 154. In general, operands may be retrieved from data storage 122 or from the micro-stack 146, processed by the ALU 148, while instructions may be fetched from instruction storage 130 by fetch logic 154 and decoded by decode logic 152. The address generation unit 142 may be used to calculate addresses based, at least in part on data contained in the registers 140. The micro-TLBs 144, 156 generally perform the function of a cache for the address translation and memory protection information bits that are preferably under the control of the operating system running on the MPU 104.

[0032] Referring now to FIG. 4, the registers 140 may include 16 registers designated as R0-R15. All registers are 32-bit registers in accordance with the preferred embodiment of the invention. Registers R0-R5 and R8-R14 may be used as general purpose ("GP") registers, thereby usable for any purpose by the programmer. Other registers, and at least one of the GP purpose registers, may be used for specific functions. For example, in addition to use as a GP register, register R5 may be used to store the base address of a portion of memory in which Java local variables may be stored when used by the current Java method. The top of the micro-stack 146 is reflected in registers R6 and R7. The top of the micro-stack has a matching address in memory pointed to by register R6. The values contained in the micro-stack are the latest updated values, while their corresponding values in memory may or may not be up to date. Register R7 provides the data value stored at the top of the micro-stack. Register R15 is used for status and control of the JSM 102. Other registers may also be provided in the JSM 102, such as one or more auxiliary registers in the decode logic 152.

[0033] Referring again to FIG. 3, as noted above, the JSM 102 is adapted to process and execute instructions from a stack-based instruction set that may include Java Bytecodes. Java Bytecodes pop, unless empty, data from and push data onto the micro-stack 146. The micro-stack 146 preferably comprises the top n entries of a larger stack that is implemented in data storage 122.

[0034] The data storage 122 generally comprises data cache ("D-cache") 124 and a data random access memory ("D-RAMset") 126. The D-RAMset (or simply "RAMset") 126 preferably comprises one "way" of the multi-way cache. Reference may be made to co-pending applications U.S. Ser. No. 09/591,537 filed Jun. 9, 2000 (atty docket TI-29884), U.S. Ser. No. 09/591,656 filed Jun. 9, 2000 (atty docket TI-29960), and U.S. Ser. No. 09/932,794 filed Aug. 17, 2001 (atty docket TI-31351), all of which are incorporated herein by reference. The stack (excluding the micro-stack 146), arrays and non-critical data may be stored in the D-cache 124, while Java local variables and associated pointers as explained below, as well as critical data and non-Java variables (e.g., C, C++) may be stored in D-RAMset 126. The instruction storage 130 may comprise instruction RAM ("I-RAMset") 132 and instruction cache ("I-cache") 134. The I-RAMset 132 may be used to store "complex" micro-sequenced Bytecodes or micro-sequences or predetermined sequences of code.

[0035] In accordance with a preferred embodiment of the invention, at least some applications executed by the JSM 102 comprise one or more methods. A "method" includes executable instructions and performs one or more functions. Other terms for "method" may include subroutines, code segments, and functions, and the term should not be used to narrow the scope of this disclosure.

[0036] A method (the "calling" method) may call another method (the "called" method). Once the called method performs its function, program control returns to the calling method. Multiple hierarchical levels of methods are possible as illustrated in FIG. 5 which illustrates the interaction between three methods (Method A, Method B, and Method C). For purposes of the example of FIG. 5, method A calls method B and method B calls method C. As such, method A is the calling method for method B which is the called method relative to method A. Similarly, method B is the calling method relative to method C which is considered the called method relative to method B.

[0037] A method may have one or more "local variables," as explained previously. Local variables may be used to temporarily store data or other information as the method performs its task(s). The local variables preferably are specific to the method to which the variables pertain. That is, method A's local variables ("LVA") are accessible generally by only method A and have meaning only to method A. Once method A completes, the method A local variables become meaningless. Similarly, LVB and LVC comprise local variables associated with methods B and C, respectively. Java Bytecodes refer to local variables using an index. The JVM maintains a local variables pointer ("PTR LV") which points to the base address of the memory containing the current method's local variables. To access a particular local variable, a suitable index value is added to the base address to obtain the address of the desired local variable. In general, the local variables associated with one method may have a different size than the local variables associated with another method.

[0038] FIG. 5 generally shows the state of the D-RAMset 126 in accordance with a time sequence of events 500, 510, and 520 as each method B and C is invoked. In sequence 500, method A is invoked and storage space 502 is allocated for its local variables (LVA). A base pointer (PTR LVA) 504 also is determined or selected to point to the base portion of LVA storage space 502. Using the pointer PTR LVA, references may be made to any local variable within method A's local variable set 502 by computing an index or offset to the PTR LVA value.

[0039] Although a plurality of methods may run on the JSM 102, typically only one method is "active" at a time having its instructions actively being executed by the JSM